Cosc460 Project
Department of Computer Science
University of Canterbury

Performance Measurement

of

Distributed Systems

Supervisor: Prof. J. P. Penny

David Tripp
October 1986

Contents

Abstract

Run time performance measurement of a certain class of distributed systems is considered, with a view to developing response time related measures for Local Area Networks. A model of user interaction with a distributed system is developed, for which *Remote response ratio* and *Remote component* are proposed as useful indices.

Monitoring tools to measure these values are described, and results are given for two Local Area Networks with different configurations and contrasting natural workloads. The lack of forethought given to performance measurement in the design of the networks surveyed and varying measures of workload is also discussed.

Acknowledgements

The author is grateful to many people who helped with this project. Thanks to Prof. Penny for much constructive criticism and direction, and to R.L. Biddle, M.A. Maclean and J. Edwards for their interest and advice.

He is also indebted to The University of Canterbury Computer Services Centre, especially E.H.B. de Lacey, and the COMRAD development team for the opportunity to measure their systems.

Final thanks are given to family and friends for their support, through prayer and encouragement, over a challenging year.

1. Introduction

"We must measure what is measurable and make measurable what cannot be measured"
- Galileo (1610)

Computer performance measurement, and its evaluation, rank with functionality and economicity as the fundamentals of a computer system [Ferr86].

Techniques for computer performance measurement are used to evaluate characteristics of a system's operation in four different situations. A system will be measured during its initial construction to aid the designer in building the most efficient system possible. When a system is being selected for a particular site, performance data is used to determine the 'best' computer for the lowest price. Once a system is installed, a performance study may be used to detect any resource bottlenecks under the natural workload, so that software or hardware changes may be made to achieve a balance between the different system units. Finally, a capacity planner would use current performance information to predict and plan for the future requirements of that user environment.

Three approaches to measuring computer performance are available, those of analytical methods, simulation, and empirical monitoring.

Empirical methods of performance evaluation are used to measure the actual operation of a real system under different conditions, such as changing workload. This can be done in three ways:

1) Specialised hardware can be connected to the system to measure, for example, traffic on a communications line.
2) Measurements, such as average response time, can be taken of the system under a controlled artificial workload.
3) Event driven or state sampling software monitors can be run on a system concurrently with its natural workload, to measure the nature of that workload and its effect on system performance.

With regard to performance, all a user is concerned about is the speed with which their requests for computer service are completed. However, measures of 'raw' response time in an interactive environment are inadequate to the performance analyst since user behaviour will affect the response time. Other indices must be defined and measured.

This need for appropriate indices is particularly important to an emerging class of computer systems, that of distributed systems. Decreasing hardware costs, increased user expectations and advances in communications technology have recently caused the rapid expansion of the number and size of distributed systems. As such systems become more complex, it is vital that performance techniques, often an afterthought in system design, keep pace with hardware developments. Otherwise performance problems will become buried in the spaghetti of large distributed systems.

This project was undertaken to investigate current performance methods and tools for a class of distributed systems, that of local area networks, and to define and evaluate any new measures that could be appropriate to the accurate diagnosis and description of performance problems.

Two factors prohibit the immediate translation of performance techniques from the traditional time sharing environment to a distributed system. In a distributed system, such as a local area network, there are many more units to monitor than in a time sharing system, rendering old techniques inadequate. As well, typically no one unit within a distributed system knows the current state of each of the other units at any one time, so a centralised approach to monitoring may no longer be possible.

Two monitoring methods have been implemented to facilitate this transition. One tool monitors the fileserver of a local area network, a potential bottleneck in the system, while the other is a distributed monitor which runs locally in each unit, with results being collected for later analysis.

In the next chapter of this report, a description of distributed systems and user behaviour is used to develop two indices for describing the performance of a local area network. Chapter 3 surveys previous methods used to evaluate the performance of local area networks. One particular network operating system is described, and then details are given of two network installations that run the operating system on which the two monitors outlined above were implemented. Chapter 4 describes these software monitors, Trimon and Netmon, in detail, with results of their operation following in Chapter 5. In Chapter 6 conclusions are drawn regarding the difficulty of monitoring the Novell Network, the value of the proposed indices, and the results of the workload monitoring of Chapter 5.

2. A Model of System Performance

A model of a system and its behaviour, on which the empirical techniques and tools of a performance study are based, can be divided into three categories. These are the submodels of the distributed system, the user behaviour, and the interaction between the user and system. These are discussed below, with particular reference to local area networks.

2.1 The Distributed System

A system where the computation is spread across more than one physical processor can be described as either tightly or loosely coupled [Pete85]. In a tightly coupled system the processors share the same memory and clock (figure 2.1).

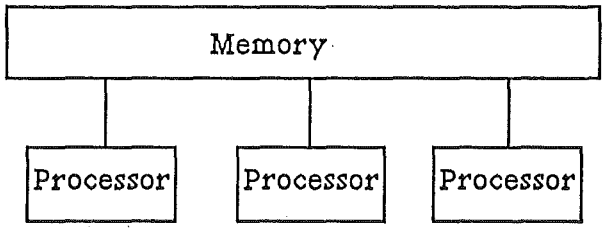

figure 2.1 A tightly coupled distributed system

In a loosely coupled system, however, each processor has its own memory, and the processors communicate through communication channels ranging from high speed buses to slow telephone connections. These are usually referred to as distributed systems. Loosely coupled systems can be configured in two ways. Communication can be achieved by the processor sharing a section of memory with the communications processors, or communication can be direct to the processor in each distributed site (figure 2.2)

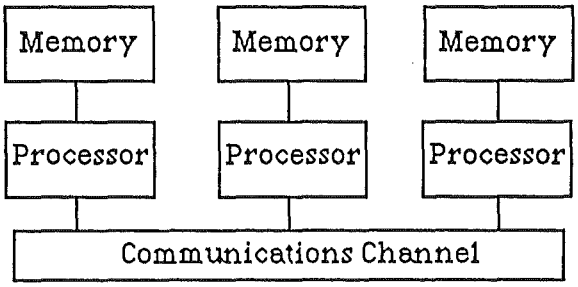

figure 2.2 A loosely coupled system

Distributed systems can be further categorised as computer networks, with sites spread over a wide geographical area, or local area networks, typically confined to a building or specific small area. The major differences are in the speed and reliability of the communications subsystem, with resulting differences in the operating system.

The main reasons for using a distributed computing approach are those of resource sharing, computational speedup and reliability.

Resource sharing allows for more expensive or infrequently used hardware or software resources to be utilised by more than one user, saving unnecessary or costly duplication.

Computational speedup may be achieved by partitioning the workload into autonomous subcomputations that can run on separate processors. For example, if the processes running on a multitasking machine were distributed over several processors, a technique called load sharing, computational speedup should result if the combined power of the processors were greater than that of the original multitasking processor.

The ability of a distributed system to continue partial operation despite the failure of one or more nodes leads to the decreased down time of the system. This, however, is dependent on the 'logical distributedness' of the system. If a site is dependent on the resources of other nodes for its normal functioning, such as access to a remote database, operation will be impaired by the failure of those remote resources regardless of the degree to which the node is physically autonomous. Hence, the characterisation of a real system by this model requires the specification of both the physical independence of a node from other distributed sites and the logical dependence on remote resources to complete that node's normal functions.

The file system and the methods of computation are the remaining two elements needed for the description of a system model.

The major issue relating to a distributed file system is the location at which the files will be stored. Two approaches are possible, the first being the use of a fileserver as a centralised solution to the storage of shared files. The location of the file is essentially transparent , with local and remote files being accessed in the same manner. The second possibility is for each site to maintain its own file system, with either the user or the operating system knowing the location of remote files that may be required.

The mode of computation is the method by which the hardware and software resources are shared by a distributed system. This divides into two categories: data migration and job migration.

Data migration occurs when a process, running on a given processor, imports data from another node of the distributed system, and makes any translations necessary for it to be compatible with the local node's requirements. Job migration involves relocating the execution of a process, either to balance the load between lightly and heavily loaded nodes, to speed up computation by subdivision across several nodes, or because of specific hardware requirements of that process.

Therefore an abstraction of a distributed system must specify the physical and logical dependence on other nodes, the manner of communication between them, and a description of the file system and methods of computation.

2.2 User Behaviour

Raghaven and Kalyanakrishnan have stated that "The characterisation of workload is an essential prerequisite to any computer systems performance evaluation study." [Ragh85] In an interactive environment, this consists of identifying the time and nature of a 'representative user session', if this can be done.

In an environment where users form natural groupings, independent of their use of the computer, this classification of users may map accurately to the computer environment. For example, in a teaching environment, students doing assignments could be classified by subject, as students within a subject would be doing similar work. In a commercial environment, classification of users into groups such as data preparation users, advanced users and secretarial users may be appropriate. This is not necessarily a classification of users by the applications they use, but by the nature, order and frequency of their use of a range of applications.

In some instances, use of system resources is largely dependent on the intensity of user activity, such as in an editing session. Therefore, classification of users according to the type of operations they execute at the functional level and the speed with which they execute interactive commands is an appropriate way to describe a model of user behaviour.

The user's level of familiarity with the system will affect his or her behaviour. An advanced user will make more extensive use of the system features and interact with the system more rapidly, resulting in a greater demand on system resources.

The number of users is frequently used as a convenient measure of workload. In the above discussion it was seen that the user's class will affect the physical demands made of the system and thus the real workload. Therefore the number of users, their class and the intensity of user activity are needed to form a more comprehensive workload index.

As performance deteriorates, long response times may affect the user's train of thought and thus their level of activity. That is, the amount of work done by the user may decrease faster than is suggested by the increase in response time [Barb83].

The time of the user session may affect performance. If users' sessions are somehow synchronized, for example by work deadlines or lecture timetables, performance will be likely to depend on the time within that period, as well as the number of users. For example, performance on a local area network would be worse when a number of users printed the results of their session and logged off than when they were all editing a document.

In summary, behaviour of individual users might be characterised by specifying each user's group, the intensity of their activity and the time of their session. Care must be taken when using simply the number of users as a measure of workload, due to the many factors that cause variations in the user's behaviour, such as poor system performance and work deadlines.

2.3 The Result of User/System Interaction

Subsequent to a system independent description of user behaviour, the final step in the development of a model of system performance is an analysis of the effect of user behaviour on the system resources, with a view to defining indices that will give a representation of run time performance.

In a distributed system, performance degradation will result from contention, that is competition with other users, for the shared resources of the system. Where the user operates a uniprogrammed node, such as in a local area network of PCs, the response

time RT(i) for some interaction *i* can be divided into two components:

$$RT(i) = RT_L(i) + RT_R(i)$$

where $RT_L(i)$ is the *local component* of interaction i, the time spent running locally,

and $RT_R(i)$ is the *remote component*, the time spent communicating through, running on, or being queued for the shared system resources.

For the above equation to hold, the remote and local states must be disjoint, that is, there must be no concurrency between running locally and serving requests from the user at a remote site. This is likely to be true for a uniprogrammed node, and is in fact true for the systems studied, because the commands of the only process running on that node are executed sequentially. If one of these commands requires service from the network, local execution of that process is suspended until the completion of that remote service.

From this the *remote component ratio* RCR(i) can be specified for an interaction i, the proportion of time for which the service of that interaction is carried out at a remote site, or, if the communication channel is shared, communicating with that remote site:

$$RCR(i) = \frac{RT_R(i)}{RT(i)}$$

For a set of *n* interactions, the *mean remote component ratio*, RCR is then:

$$RCR = \frac{RT_R}{RT}$$

where

$$RT_R = \frac{1}{n} \sum_{i=1,n} RT_R(i) \quad \text{and}$$

$$RT = \frac{1}{n} \sum_{i=1,n} RT(i)$$

Similarly the *mean local component ratio*, LCR, can be defined as:

$$LCR = \frac{RT_L}{RT}$$

where

$$RT_L = \frac{1}{n} \sum_{i=1,n} RT_L(i)$$

If any interaction is repeated, $RT_L(i)$ will be the same for the second interaction, as there is no contention for local resources in a uniprogrammed environment. Note also that, provided the remote and local states are disjoint and are the only possible states in which a user's process can be found, then:

$$LCR + RCR = 1$$

The degradation in service time from the system resources can be measured by the *response ratio* RR(i), being the ratio of the actual response time to the minimum possible response time for that interaction [Penn84]. For interaction *i* :

$$RR(i) = \frac{MT(i) + WT(i)}{MT(i)}$$

where MT(i) is the minimum possible response time for interaction i, and
WT(i) is the time additional to the minimum to complete an interaction on a loaded system.

If MT(i) and WT(i) are broken into their remote and local components then RR(i) becomes:

$$RR(i) = \frac{MT_L(i) + WT_L(i) + MT_R(i) + WT_R(i)}{MT(i)}$$

$$= \frac{RT_L(i)}{MT(i)} + \frac{MT_R(i) + WT_R(i)}{MT(i)}$$

since $WT_L(i) = 0$ and $MT_L(i) = RT_L(i)$ on a uniprogrammed network node.

The response ratio RR can be determined in terms of the remote component, the local component and the remote response ratio. The *remote response ratio* $RR_R(i)$ is the ratio of the actual service time of a request using only the shared resources to the minimum time for that request, that is:

$$RR_R(i) = \frac{MT_R(i) + WT_R(i)}{MT_R(i)}$$

then

$$RR(i) = \frac{RT_L(i)}{MT(i)} + \frac{RR_R(i) * MT_R(i)}{MT(i)}$$

and MT(i) can therefore be expressed as:

$$MT(i) = RT_L(i) + \frac{RT_R(i)}{RR_R(i)}$$

RR, the response ratio of a set of n interactions is then:

$$RR = \frac{\Sigma (RT_L(i) + RR_R * MT_R(i))}{\Sigma MT(i)} \qquad [1]$$

where RR_R is defined as

$$RR_R = \frac{\Sigma (MT_R(i) + WT_R(i))}{\Sigma MT_R(i)}$$

These measures, RR and RR_R, are useful in several ways. Suppose that a shared hardware unit is upgraded. One can calculate the improvement expected in the performance for each user by substituting the anticipated remote response ratio in equation [1]. Measurements of RR taken after the change can be used to validate the extent of the improvement.

An upper bound on the expected percentage performance improvement is given by:

$$RT = RT_L + \frac{RT_R}{IF}$$

where IF is the *improvement factor*, the ratio of the speed of the shared resources after upgrading, over the speed before upgrading.

This result is important. It is shown in chapter 5 that substantial improvements to the fileserver of the Undergraduate Network would make only a very small improvement in the performance experienced by users.

If a set of interactions using one application are measured, the demands that application makes of the shared resources can be calculated and used by the capacity planner to anticipate the effects of increasing workload or the addition of further nodes to the distributed system.

The level of activity of a user with respect to the shared resources can be established by examining the RCR values for each user running the same interactive application. The average RCR for each user could be used as an indication of their relative activity compared with other users.

3. Background to the Measurement of LANs

Evaluation techniques used to measure the performance of a subclass of the model of distributed systems described in section 2.1, that of Local Area Networks, are outlined below. Sections 3.3 and 3.4 describe a specific LAN operating system and a comparison of the two LAN installations studied.

3.1 Previous Work

A number of different methods have been used to measure the performance of distributed systems. A common approach is to analyse the communication links between the nodes of a network.

For example, Hammond and O'Reily [Hamm86] developed quantitative performance measures for a local area network. At the user to user level (application to application layer in OSI nomenclature) mean response time, standard deviation of response time, and the percentage of response times that exceed a fixed value are proposed as performance measures. At a network level, throughput, in bits per second, average delay at the network level and channel utilization are suggested as suitable measurement indices. These, however, are all defined in terms of transmission times over the communication network connecting the nodes.

The approach taken in this report is more global. The time an interaction spends in the communications subsystem is only one component of the total response time, and hence only one place in the system where delay can occur.

Rajarman [Raja84] defines a number of parameters that describe the performance of a distributed system. These parameters are grouped under three categories: those related to job characteristics (such as resource requirements and priority), to operational characteristics (parameters set by the system administrator) and to the network interface (activity on the network and the number and type of users). Using these parameters, Rajarman defines four types of indices that describe the system's performance:

1. System performance measures (e.g. job delay and average productive time)
2. System component utilisation (e.g. CPU or channel utilization)
3. System interface efficiency measures (e.g. file transfer efficiency)
4. System load measures (e.g. the percentage of jobs belonging to each class)

These indices were developed for a network of 3 large mainframes and an array processor in a batch-orientated environment. Many of the indices are throughput related and therefore not relevant to an interactive environment. However, some measures of type 2 and 3 are still applicable.

Balkovich and Soceanu [Balk82] evaluated the performance of a LAN running programmes written in EPL, an experimental programming language for distributed computing. Evaluation was made possible by linking each node with a second, independent network to transmit performance information to a host device.

Hays [Hays85] has suggested that five hardware orientated features determine the performance of PC LAN's, such as those used as experimental sites for this report. These are:

1. The design of the hard disc, including the access time of the read/write head and the head position technique used.
2. The sector interleave factor of the disc controller.
3. The type of the central processor.
4. The operating system.
5. The communications processor and protocol.

Monitoring of the University of Canterbury Undergraduate Network demonstrated that the size of the fileserver memory was also significant.

The use of artificial methods, including scripts and benchmarks, provides reproducible results of a required accuracy in a much shorter time than by monitoring the system's natural workload. However, Ferrari [Ferr84] points out that design methods for artificial workloads often ignore the dynamics of the workload being modelled and that the resource orientation of a test is often unrelated to the demands on that resource under the natural workload.

3.2 The Use of Benchmarks

The most common technique found to be used for the performance evaluation of commercial PC LAN systems is benchmark testing of the IO subsystem of the fileserver. Benchmark comparisons between different hardware and operating system configurations are common in PC literature. These benchmarks often consist of measuring the average time to read, write or create and delete files of varying sizes, for different number of concurrent 'users'.

Unfortunately, benchmark results have limitations, as outlined by Buzen [Buze76]. Benchmarks are of some use, however, in showing which hardware/software configurations are 'useless' and the relative difference in speed between isolated hardware units.

Inaccuracy in the use of benchmarks arises from the difference between the benchmark test and the real workload. The ideal method of system selection would be to compare different configurations under the actual user workload, and from this determine the balance between different hardware resources required by the real workload. For example, a system that is known to make little use of the shared IO resource could tolerate a slower fileserver for greatly reduced cost.

Benchmark results only reveal the capacity of the bottleneck in a system under that test. A workload of a different nature or a different network configuration may result in fewer demands on that system resource, and another resource becoming the bottleneck. Only when the bottlenecks during benchmark testing were the same as the bottlenecks occurring under the real workload could benchmark data be accurately used to compare different hardware configurations.

It is often argued that as most fileservers only perform one main function (disc IO) that benchmarks reveal the best configuration for system selection. However, this function is carried out by different hardware and software resources, each of which could be introducing delays of varying lengths into the response time for an interaction.

Neither does the typical benchmark reveal what the bottleneck is. An example is documented [Step86] where a benchmark result was improved by over 100%, as a result of the workstation-fileserver communication protocol being upgraded from a 'stop and wait' protocol to a more efficient means of data transmission. The benchmark was testing fileserver IO speed.

As fileserver architecture becomes increasingly complicated, the confidence with which benchmark data can be used to extrapolate approximate performance under the real workload must decrease. The current trend is towards mutlitasking fileservers where the disc, processor and communications subsystems could each be the bottleneck under different workloads. As network configurations become more complex, with the introduction of multi-fileserver networks and inter-network bridges, the accuracy of performance information determined using an artificial workload will decrease.

For system selection, benchmark data may have to be relied upon as evaluation of a new system under the real workload, before installation, would be likely to be impractical. Performance monitoring of an existing system, whether for reasons of capacity planning or bottleneck detection, is, however, more accurately undertaken by monitoring the effect of the real workload on the system resources, rather than relying on methods using artificial workloads, such as benchmarks.

3.3 The Novell Network

The network studied for this report is described below, and the anticipated influence of the network's software and hardware on the network's performance is considered.

The Novell Netware and Advanced Netware operating systems network IBM PCs and IBM compatibles to a fileserver and other site-dependent shared resources. Advanced Netware allows multiple fileservers on one network and inter-network connection between fileservers to form clustered larger networks. Facilities such as user security (login, different user groups), file security (trustee rights to directories), print spooling, inter-workstation signalling and communication, and electronic mail are supported.

Some hardware configurations allow the fileserver to be non-dedicated, that is, it is also able to be used as a workstation. Workstation-fileserver communication is by twisted pair, or baseband or broadband coaxial cable.

The Netware LAN environment is modelled on the ISO Open Systems Interconnection seven layer model as follows:

Application layer	- Netware Utilities, user applications
Presentation layer	- DOS, Netware Shell, File Server
Session layer	- Netware shell, file server
Transport layer	- & NETBIOS emulator
Network layer	- IPX
Link layer	- Hardware dependent
Physical layer	- Hardware dependent

Under Netware the Physical and Data Link layers are provided by the hardware, the protocols varying from manufacturer to manufacturer. The Network layer is controlled by Netware's Internetwork Packet Exchange Protocol (IPX) which controls inter-network routing and communication.

The workstation utilises the shared network resources in the following way (figure 3.1). All requests to the operating system from an application running in a workstation are processed by the Netware shell, which intercepts calls to DOS in each workstation. If it is a local request it is passed on to DOS, otherwise it is encoded in a packet and sent to the fileserver via the workstation's network communication driver. The reply returns through the communications driver to the application.

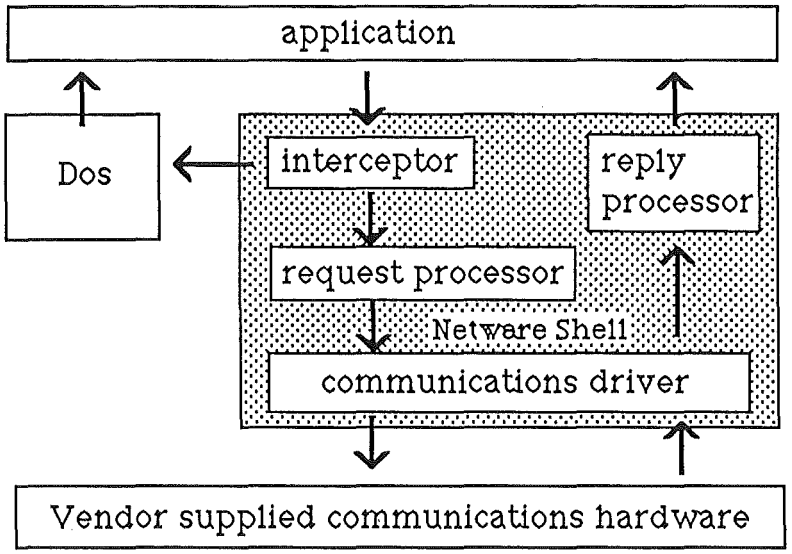

figure 3.1 The workstation Netware shell

The multitasking kernel of the fileserver provides services which include system administration functions via the console, print spooling, network communications and file handling (figure 3.2). Record locking and other measures to ensure shared data integrity are supported by the disc process.

The efficiency of the disc process is enhanced by directories being hashed to avoid sequential directory searches, and cached in memory to decrease disc accesses for directory searches. Any spare memory is used for file caching on a Most Recently Used basis, retaining frequently accessed files in memory. When a file is written to disc it is cached, memory permitting, until the fileserver is idle; it is then written to disc. An elevator-seeking algorithm is used by the read/write head to increase the transfer rate for multiple disc accesses.

The spool process spools files to the disc. The files are then sent sequentially to the network printer(s).

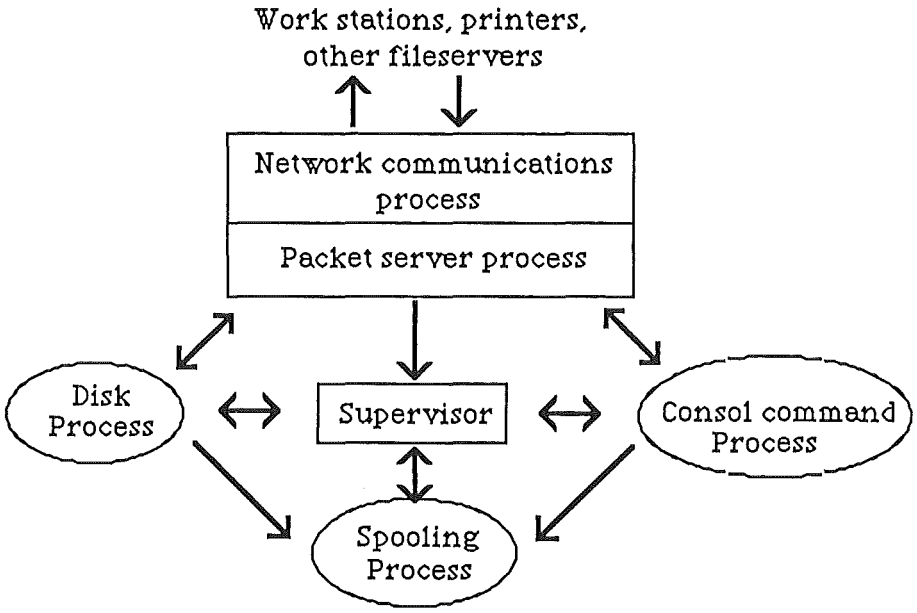

figure 3.2 The fileserver operating system

The 'black box' nature of the fileserver, on which no user written software could be run, the absence in New Zealand of any expert on the Novell Network, and the lack of any source code for the operating system or any technical documentation caused considerable frustration in attempting to monitor networks using this operating system.

3.4 Two Network Sites

Performance data was obtained from two systems, the University of Canterbury Undergraduate Network and the Network in the Radiology clinic at St. George's Hospital. Specifications for these two distributed systems are contained in Appendix A.

The University of Canterbury Undergraduate Network was installed to replace the previous student computing facility based on an aging PDP 11-34. The network is used by students doing class assignments in computer science, operations research and mathematics. The network consists of two clusters connected by an inter-network bridge. Each cluster has 20 PC workstations, 2 printers and a Novell fileserver.

COMRAD (Computer Orientated Management of Radiology Department) is a system being jointly developed by the Christchurch Hospital Board and the private radiology clinics in Christchurch and Nelson. The author was involved in software development over the '85-'86 summer vacation. The functions of the system are to maintain medical records for all patients (40,000 a year) visiting the Radiology Department of

Christchurch Hospital, with up to 5 years of visits online, to produce reports of visits to be sent to referring doctors, and to provide statistical and accounting information for management purposes.

St. George's Hospital is being used as a trial site for COMRAD. Four PC workstations are connected by a multidrop cable to an IBM AT fileserver.

The workstations and network operating system are similar for both systems under consideration. The major differences relevant to this study are the network configuration and the applications used.

The differences in configuration that will affect performance are the fileservers and the method of linking the workstations. Different hardware is used for each fileserver. Although the operating system is very similar, differences in clock speed, processor, memory size and disc controllers will affect the relative performance of each system.

The workstations of each cluster on the Undergradate Network are all connected to the fileserver by a separate communications line. Since each workstation is a uniprogrammed system, each line will never be used for more than one task at any one time. Therefore there is no contention, and thus no delay, in the workstation to local fileserver communication. All users logged in remotely from one fileserver to the other communicate with the remote fileserver over one channel. Transmission delays would result from two or more remotely logged on users concurrently accessing their private files (public files are stored on both fileservers to avoid congestion on this link). The inter-network bridge and the hardware and software components of the fileserver are the only shared resources of the system.

The Radiology network, however, uses one multidrop line over which each workstation communicates with the fileserver. If more than one workstation attempts to transmit concurrently, there will be contention, and delays will be incurred. The whole communications subsystem is therefore a shared resource.

The second performance-related difference between the two sites is the resource requirements of the applications used. The Undergradate Network is primarily used for program development by students. Editing, compiling and running small programs results in a high user think time. The workload is CPU intensive and, as each workstation has a local processor, therefore makes few demands of the shared resources of the system.

The Radiology network is used to maintain a large database. Competition for the shared resources of the IO subsystem will therefore be a larger component of the average response time. It is anticipated that this will cause more rapid degradation of response time as the number of users increases, compared with the Undergraduate Network.

4. Measurement Techniques

The following two sections describe the two measurement techniques implemented on the Novell Network. Results obtained from these methods are discussed in Chapter 5.

4.1 Determination of Remote Response Ratio

Response ratio for service from the shared resources of a system is defined as the actual response time for that service divided by some value that represents the inherent complexity of the request made [Penn84]. In the monitor described below the optimal or minimum response time is used as a measurement of the complexity. It is assumed that the optimal response time is the response time when no other users are logged onto the system.

To calculate the response ratio of the fileserver, a 'synthetic job' approach is used [Penn80]. The monitor, developed to measure the remote response ratio of the fileserver, runs on a workstation, having the same priority as all other users, and continually repeats the same IO function. The time taken for this sample, relative to other samples, is used as a measure of the contention for the shared disc resource. The average time for samples taken when there were found to be u users logged on, divided by the average time for this IO function when only the monitor was running, gives a response ratio for service by the file server when u users are logged on.

Although this technique is essentially one of recording the time of an artificial job, the IO function is used to measure the real workload and so the inaccuracies of benchmarks outlined in section 3.2 are not possible. This method is, however, potentially disruptive to a systems operation due to the marked increase in total workload.

Possible inaccuracy in this method is caused by the structure of the network operating system. Disc IO is only one of the three multi-tasked functions of the operating system. Requests for service from the communications or spooler subsystems will compete for some, but not all, of the fileserver resources that the IO function utilizes. The results will only be valid if the bottlenecks for disc IO are common to all functions of the fileserver. Thus using disc IO as an indication of the response ratio for service from the whole fileserver requires that disc IO be the predominant service required by users. On both systems under examination this is the case as inter-workstation communication is minimal and spooling is disc based, so this utility must also contend for disc access.

The monitor , written in C, is invoked with the command line:

trimon [-mup]

the argument *m* runs the response monitoring as described above. *p* and *u* monitor the average number of users for each 5 minute period. *p* prints the information to the screen, *u* writes it to a file whose name is requested from the user.

The results of the *m* option are written regularly to a file specified by the user. These are stored in an array of U_{max} rows, with the u^{th} row being:

$$\Sigma RT_{R,u}(i), \quad \Sigma (RT_{R,u}(i))^2, \quad N_u$$

where:

U_{max} is the maximum number of users possible

$RT_{R,u}(i)$ is the remote response time for the i^{th} sample for which there were found to be u users logged on

N_u is the number of samples for which there were found to be u users.

The mean remote response time for *u* users, $RT_{R,u}$, is given by:

$$\frac{1}{N_u} \Sigma RT_{R,u}(i)$$

and the variance for u users given by:

$$\frac{1}{N_u} \Sigma (RT_{R,u}(i))2 \quad - \quad \frac{(\Sigma RT_{R,u}(i))^2}{N_u}$$

Execution of this monitor on the Undergrad Network was frustrated for a time by network errors, which required the user to repond to a restart query. These become more frequent as the workload on the fileserver increased. The number of such network errors decreased markedly after the addition of further memory to the fileserver.

Achieving sufficient accuracy in the timing of the IO job was difficult, the method used is described in Appendix B. The error in timing was reduced to ±0.006 seconds. The IO job done by this monitor consists of repeatedly opening, writing 100 characters to, and closing 5 files.

The number of users currently logged onto the local fileserver is obtained using a software interrupt (e3h, subfunction 05 of interrupt 21h) which returns login information about each workstation, including the usercode, login time and date. This call is

repeated for each possible workstation, a logged in user is detected by the existence of a user code. Users logged to the remote file server from the local file server have the user code REMDUM on the local server and so can be distinguished from 'local' users. Users logged on to the local file server have a station number less than 25, and users logged on to the local file server from the remote server have a station number greater than or equal to 25. These values are specific to the Undergraduate Network and were used to produce user/time graphs, an example of which is listed in Appendix C.

4.2 Determination of Remote and Local Component

The remote component ratio is the percentage of time that a network node is waiting on a reply to a request for network service. The local component ratio is defined as the average percentage of time that a node is running locally or in user think state. Both these measures will vary depending on the type of application being used, and, if in an interactive mode, the level of user activity. The monitor described below was used to determine the average remote and local components of response time for a user session, and to investigate the effect of network workload on these measures.

Knowing the average remote component of an application is important if the capacity planner is considering the addition of further nodes to the network. The number of workstations that can be supported, within a given level of performance degradation from the shared resources, can be extrapolated from these measures using queuing theory. Obviously the performance would be more severely effected by the addition of workstations to a network where the remote component ratio of each node was 50%, than the addition of workstations where each node had a remote component ratio of 2%.

Due to the multitasking nature of the fileserver and the concurrent workstation/fileserver transmission possible on a star network, it is feasible for the sum of the remote components of all nodes on the network to exceed 100%, although service from the shared resources would be likely to be very poor under such circumstances.

The local and remote components are found using a state sampling monitor, Netmon, which runs locally in each workstation. Netmon is a clock driven, memory resident program written in Intel 8088 assembler. The code, when run, copies the monitoring procedure into the code's program prefix segment in memory, where it remains resident regardless of other user activity. The starting address of the monitoring procedure is loaded into an interrupt vector provided for user service routines. The monitor is

subsequently called by BIOS, as a result of a hardware clock interrupt, 18.2 times a second.

Ferrari [Ferr83] states that the accuracy of the results of a sampling monitor depend on the observations being independent of the time instants at which samples were taken. Independence is not satisfied if:

1) The observed phenomena are somehow synchronized with the sampling interval, or
2) An uninterruptable function is being executed when the interrupt occurs.

Neither of these conditions will occur with Netmon. This monitor is driven off the clock interrupt which is the highest of eight prioritized levels of interrupt [IBM84].

A state sampling monitor should not significantly slow the operation of the users applications. For benchmarks run on the Undergraduate Network, Netmon slowed the CPU running time of a CPU intensive function by an average of 0.13%, which is considered acceptable.

The monitor routine determines whether the workstation is waiting for a reply to a network request by finding the value of the program counter before interruption, saved on the stack as a return address (figure 4.1).

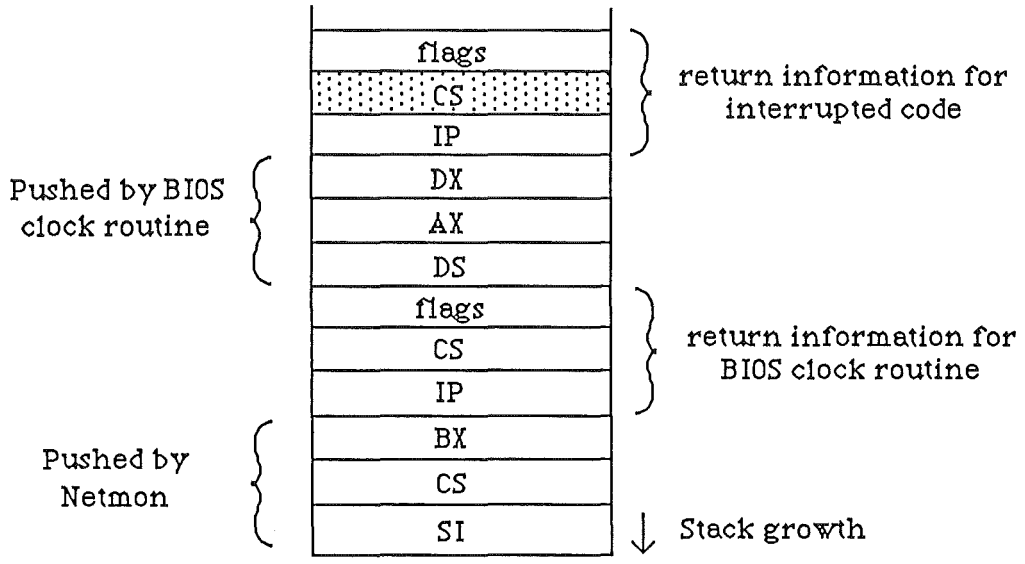

figure 4.1 Return addresses on the stack

Experimentation using the code listed in Appendix D.2.5 revealed the same value in the Code Segment register whenever the workstation was waiting for a network reply. This

value, loaded off the stack, is used to test whether the workstation is in 'local' or 'remote' state. Division of the 'local' state into 'running locally' and 'think state' would have been a valuable refinement as this could then have been used as a measure of an interactive user's activity. This was not possible as no way could be found to determine whether the PC is waiting for keyboard input from the program counter.

The count of the number of times the workstation was found in each state is kept in an unused area of low memory.

The assumption is made that when the workstation is waiting for a reply from the fileserver, the remote resources being used are shared resources. On a star network this is not the case as each user has a private communication channel to the fileserver, which Netmon considers a remote resource. This channel, however, has a much higher data transfer rate than the disc subsystem, so will only account for a very small proportion of the remote component.

Some applications, such as Sidekick (a memory resident editor and note pad), use the interrupt described above. No application has been found which interferes with Netmon's operation, but there is the potential for a user application to disconnect Netmon from the clock interrupt by taking over the interrupt itself and overwriting Netmon's call address.

Netmon is a distributed monitoring scheme. Results are collected in each workstation and written, when the user logs out, to a common file for later analysis. This monitoring suite consists of 4 modules, listed in Appendix D.2:

1) Netmon itself is loaded by the Autoexec.bat file, which is run when a workstation is booted.
2) A program called zero is run from a users' login script. This sets the state counters to zero so that only the logged in period is recorded.
3) Netread is run from the logout batch file. This reads the counters from memory, finds the number of users on the system and the user's user code and writes this information to a result file.
4) Netanal analyses the result file, writing out the average remote component ratio for each number of users found to be logged on when a workstation logged off.

A more accurate method for comparing the remote and local components to a measure of the workload would involve counting the number of users when each sample was taken,

and adding that sample to a table recording the number of samples for each number of users found logged on when the sample was taken. This was not possible as determining the number of logged on users involves using fileserver functions taking about 2 seconds to run. It could not, therefore, be repeated 18 times a second. This approximation is acceptable, since the number of users on the system changes slowly, as is seen by the user/time graph in Appendix C.1.

The output of Netanal is a list of U_{max} rows, the u^{th} row being:

$$u, \quad N_u, \quad RCR_u, \quad \Sigma RT_{L,u}(j), \quad \Sigma RT_{R,u}(j)$$

where:

U_{max} is the maximum number of users possible

N_u is the number of user sessions for which there were found to be u users logged on as a user logged off.

$RT_{L,u}(j)$ is the number of times the workstation was found in local state for the j^{th} user session for which there were u users logged on as this user logged off.

$RT_{R,u}(j)$ is the number of times the workstation was found in remote state for the j^{th} user session for which there were u users logged on as this user logged off.

$RT_{R,u}$ is remote component ratio, the average percentage of remote to total states for which there were found to be u users logged on as a user logged off.

The remote component ratio for *u* users, RCR_u, is given by:

$$\frac{\Sigma\ RT_{R,u}(j)}{\Sigma\ (\ RT_{R,u}(j) + RT_{L,u}(j)\ )}$$

5. Results

5.1 Workload Characterisation

The workload generated by student use of the Undergraduate Network distinguishes it markedly from a commercial or database system. On this system, typically only one application is being used at a given time by most users, this application changes from assignment to assignment. For example, Turbo Pascal is used by 400 students doing a Cosc 111 assignment, the Cosc-1 assembler by 200 students doing Cosc112 and so on. Applications such as these tend to be memory based, therefore making minimal demands on the file server. Other applications such as PC-SAS and Open Access are available but not used by whole classes.

Thus at any given time the workload is of a fairly homogeneous nature, and results from monitoring under these conditions would be applicable only to that application. Monitoring under a more heterogeneous workload would allow results of monitoring over a given period to be compared with any period, for example after a system change. On the Undergraduate Network comparisons are only valid for periods using the same application.

Monitoring of the Undergraduate Network using Trimon revealed several characteristics of user behaviour. An example of the output is listed in Appendix C.1, results for which were collected over an 18 hour period, 6 working days before a Cosc111 compulsory assignment was due. It can be seen that the number of users peaks during mid afternoon, decreases markedly over lunch and dinner times, and drops slightly on the hour, as students leave for classes.

5.2 Evaluation of a System Change

In July 1986 the main Undergraduate fileserver memory was expanded from one to four megabytes in a successful attempt to reduce the number of network errors. Measurement of the remote response ratio before and after the change showed a reduction in the mean remote response time. The monitor was completed just prior to this upgrade, so only a short period could be monitored to obtain the 'before' results (listed in Appendix C.2).

Using the method described in section 4.1 the remote response ratio of service by the fileserver was calculated as a function of the number of users. The monitoring was done when a stage one programming assignment was current. System utilities, such as login, logout, and directory listings were used by all users.

Due to the greatly variable length of time taken by the IO job, a large number of samples were required for each number of users. Values for which less than 100 samples were obtained were discarded. A complication was caused by remotely logged on users accessing public utilities on the remote server, rather than the same utilities on the local server.

The minimum time for the IO job, MT, was taken as the average response time when one user (the monitor) was logged on, and is given by:

$$\frac{\Sigma RT_{R,u=1}(i)}{N_1}$$

Thus the remote response ratio for each number of users, $RR_{R,u}$, is given by:

$$\frac{RT_{R,u}}{MT}$$

This value indicates the relationship between the expected time and the minimum possible time for disk service for that number of users.

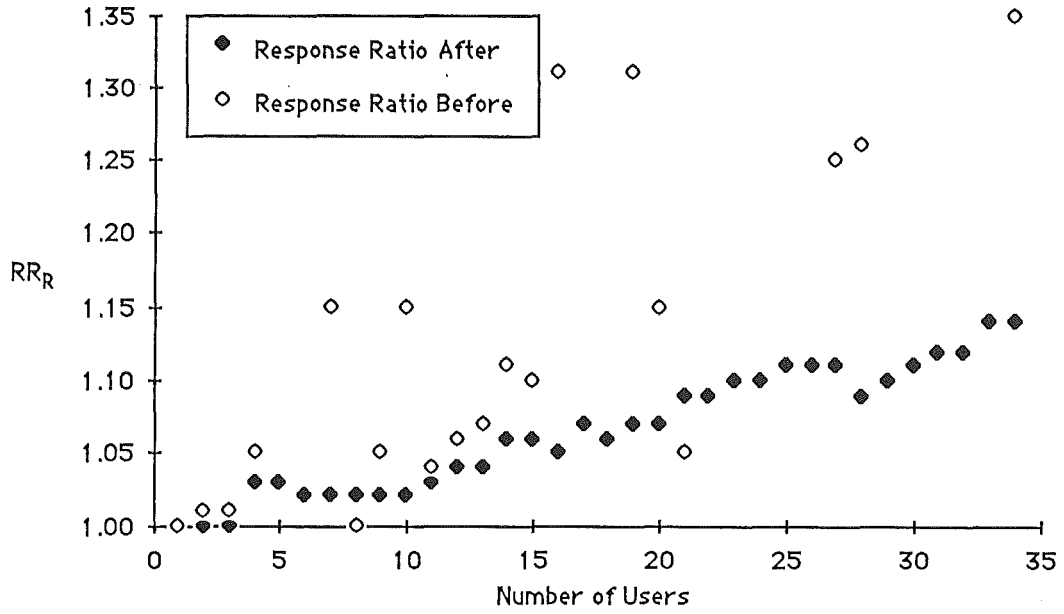

figure 5.1 Remote response ratio before & after system change

Figure 5.1 shows this relationship before and after the system change. The 'before' curve is relatively inconsistent due to the small number of samples taken. It is seen that a marginal improvement in performance resulted from the upgrade. However, the degradation in the service time for the remote component is minimal, for increasing numbers of users, and thus this improvement is insignificant in terms of a whole session, as is shown below.

5.3 Evaluation of Response Ratio

Using the state sampling method described in section 4.2 the remote and local components were measured. The results of this monitoring are listed in Appendix C.3. Monitoring was undertaken over a month with a predominantly small number of users and a peak caused by a stage one programming assignment.

The samples were compiled into a table against the number of users on local workstations, as only the local workstations were monitored. The mean remote component ratio was calculated and is graphed against the number of users in figure 5.2.

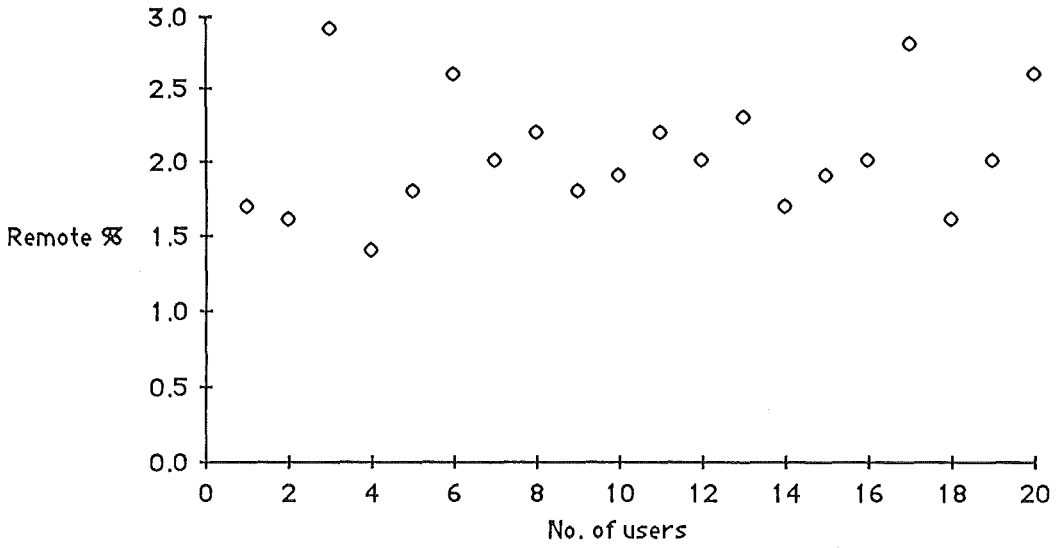

figure 5.2 Remote component ratio vs Number of users

From these results it can be seen that there is little correlation between the remote component ratio and the network workload, as expressed by the number of local users.

If, however, the output of Netmon is analysed by finding the average remote component ratio for all the terminal sessions of each individual user, a different picture emerges. These results are listed in Appendix C.4 for all users whose recorded login time exceeded 5 hours. From these results, it can be seen that in the absence of any substantial performance degradation the remote component ratio varies markedly from user to user. Hence, in this situation, the user's activity has a greater effect on the remote component ratio, than does the degradation in remote response ratio.

The remote component ratio obtained can be validated by deriving a theoretical value, using an M/M/1 queue, for this ratio from the remote response ratio (section 5.2). This approximation is based on the following assumptions:

i) The service time for an IO job is independent of the number of other IO jobs queued at the time. This is generally true when the number of queued jobs is small. As the number of queued jobs increased, disc access strategies increase the transfer efficiency and decrease the average service time.
ii) The arrival rate and service time for each user's IO is independent of other users, and is exponentially distributed. This is not strictly correct as disc activity increases on the hour as users log on, or print their files and log off.
iii) The fileserver is considered as one abstract process; multi-tasking abilities do not have an effect on the single server queuing formulas used.

$$P(t_q > T) = e^{-[(1-\rho)T/E(t_s)]} \qquad \text{[Mart72]}$$

$$= \text{Prob (queuing and serve time } t_q \text{ exceeds T)}$$

Average response ratio = 1.1 for 20 users. Thus for every 1 second of IO, 0.1 seconds is spent in a non-optimal state, such as queuing.

$T = 1.1$, the average queuing and service time

$E(t_s) = 1$

$P(t_q > T) = 0.5$ to find mean utilisation

therefore:

$$0.5 = e^{-(1-\rho)*1.1/1}$$

$\rho = 0.36$ for 20 users

$\rho = 0.018$ per user.

This is of an equivalent order of magnitude to the value obtained from Netmon.

Having calculated the remote response ratio, and the remote and local components, the response ratio for the 'average' user session can be calculated using the formulae derived in section 2.3, and numerical values listed in Appendix C.3.

For 20 users:

$$MT = RT_L + \frac{RT_R}{RR_R}$$

$$= 476263 + 12729 / 1.07$$
$$= 488159$$

$$RR = \frac{RT_L + RT_R}{MT}$$

$$= (4765263 + 12729) / 488159$$
$$= 1.002$$

Thus on the Undergraduate Network under the conditions described, the performance degradation over a whole terminal session due to contention for the shared resources is insignificant.

The response ratio for each number of users can be used to make an interesting comparison with a traditional time sharing system. An earlier study by the author of the University of Canterbury Computer Science Department's Vax 750 found a response ratio curve, graphed in figure 5.3 against the results for the Undergraduate Network.

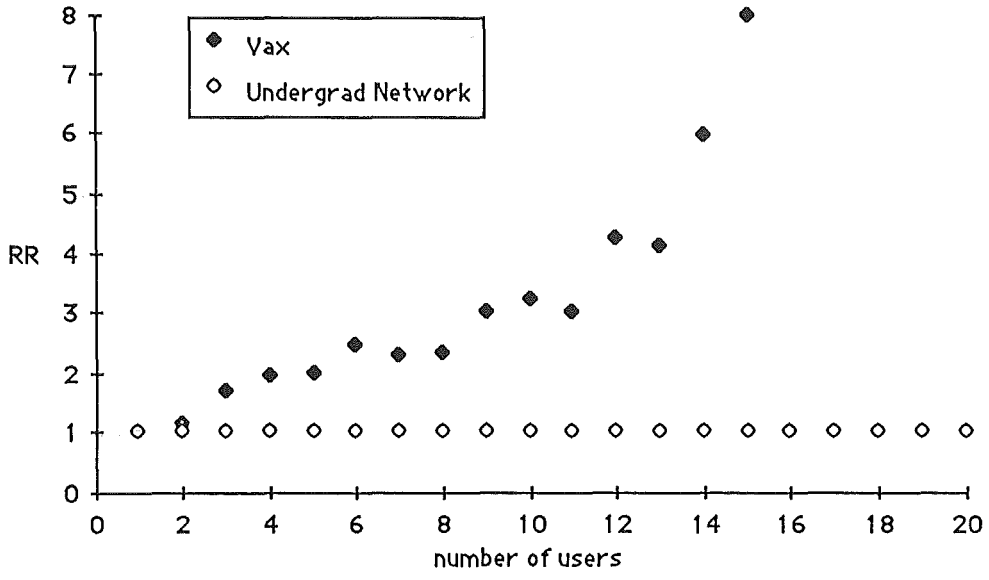

figure 5.3 The response ratio of the Vax vs Undergrad

No conclusion may be drawn from this comparison about the relative power of these two systems because the workloads are different, but the advantage of distributed processing in maintaining consistent performance in a CPU intensive environment is highlighted.

5.4 Comparison of Two Systems

It was hoped that a comparison of the Undergraduate Network and the Radiology Network could be undertaken, and that this comparison would show interesting differences between the sizes of the remote component ratios and the remote response ratios for similar systems using different applications. Unfortunately the final implementation date for the trial COMRAD system at St. Georges Hospital was repeatedly delayed, eventually being postponed until after the completion of this report. Results from monitoring undertaken during system testing were inadequate for a valid comparison.

As discussed in section 3.4, the IO intensive nature of the applications used on this system should result in a greater remote component ratio than that found on the Undergraduate Network. The increased competition for the shared network resources should also result in a higher remote response ratio. From these values, the effect on system performance of the addition of further nodes to the network could be predicted. This is an important consideration as it is planned to install COMRAD systems in larger sites where many more workstations would be required.

Also, if response times are found to be unsatisfactorily high, the effect of upgrading either the shared or local resources could be estimated.

6. Conclusion

"The insularity of performance evaluation has very often caused measurement to be an after thought in system design." [Ferr86]

Despite the importance of performance measurement to system design, selection, tuning, and planning, the systems considered lacked sufficient facilities to enable accurate and complete run time measurement of the resource utilisation and the degradation of response time with increasing workload. For example, a Novel technical manual suggests upgrading a fileserver's IO subsystem if "the red light is on a lot" [Nove86]. Existing techniques, such as the use of benchmarks, are inadequate for the task of measuring the performance of increasingly complex local area networks, and distributed systems in general.

The division of response time into local and remote components is considered a successful attempt to define further measurement indices useful in a distributed processing environment. For a uniprogrammed node, the ability to evaluate the remote component ratio, and the degradation of response time from the shared resources with increasing workload, enables the pinpointing of performance problems and the situations in which they occur. Also, the response ratio as experienced by the user can be accurately derived from these measures.

This work could be extended in two ways. Further development of Netmon to divide the local component into the 'think' and 'running locally' states would be advantageous in an interactive environment, as the intensity of user activity causes changes in the workload, which affect system performance.

Secondly, extension of the model developed in section 2 to a network where the nodes are multitasking machines would be useful, as local area networks are increasingly becoming networks of multiprogrammed nodes.

7. References:

[Balk82] Balkovich, E. E. & Soceanu, A., 'A Local Computer Network for Performance Measurement of Locally Distributed Software', *Local Computer Networks*, North-Holland, 1982.

[Barb83] Barber, R. E. & Lucas, C. L., 'System Response Time, Operator Productivity, and Job Satisfaction', Communications of the ACM, **26**:11, 1983.

[Buze76] Buzen, J. P., 'Fundamental Laws of Computer Systems,' Acta Informatica, **7**, 1976.

[Ferr83] Ferrari, D., Serazzi G. & Zeigner A., *Measurement and Tuning of Computer Systems*, Prentice-Hall, 1983.

[Ferr84] Ferrari, D., 'On the Foundations of Artificial Workload Design', Performance Evaluation Review, **12**:3, ACM Sigmetrics, August 1984.

[Ferr86] Ferrari, D., 'Considerations on the Insularity of Performance Evaluation', IEEE Transactions on Software Engineering, **12**:6, 1986

[Hamm86] Hammond, J. L., & O'Reilly, P. J. P., *Performance Analysis of Local Computer Networks*, Addison-Weslsy, 1986.

[Hays85] Hays, D., 'Fileservers: a Key to LAN Performance', Gateways Technical Bulletin #4, 1985.

[IBM84] *IBM PC XT and Portable PC Technical Reference Manual.*, Revised edition, 1984.

[Mart72] Martin, J., *Systems Analysis for Data Transmission*, Prentice-Hall, 1972.

[Nove86] *LAN Evaluation Report,* Novell Incorporated, Utah, 1986.

[Penn80] Penny, J. P. & Sheedy, C. R., 'Measurement of Response Time Performance in Small Time-sharing Systems', The Australian Computer Journal, **12**:1, 1980.

[Penn84] Penny, J. P. & Ashton, P. J., 'Measurement and description of Time-sharing System Response', Computer Performance, **5**:3, 1984.

[Pete85] Peterson, J. L. & Silberschatz, A., *Operating System Concepts*, 2nd edition, Addison-Wesley, 1985.

[Ragh85] Raghaven, S. V. & Kalyanakrishan, R., 'On The Classification Of Interactive Users Based on User Behaviour Indices', Performance Evaluation Review, **13**:2, ACM Sigmetrics, August 1985.

[Raja84] Rajarman, M. K., 'Performance Measures for a Local Network', Performance Evaluation Review, **12**:2, ACM Sigmetrics, 1984.

[Step86] Stephenson, L. J., 'Benchmarks Revealed', Gateways Technical Bulletin #8, 1986.

Appendix A: Network Specifications

The Undergraduate Network.

Hardware Specifications:

The system is made up of 2 clusters. Each has a Novell fileserver and console, the processor being a MC68000. During the course of this study the memory was expanded from 1 Mbyte in each fileserver to 4 Mbytes in one and 2 Mbytes in the other. The two clusters are connected by inter-network bridging software and a communication link. Each cluster is a star network, with each workstation having its own line to the fileserver, using a protocol similar to RS442, at approximately 600 Kbits per second. Each fileserver has one 50 Mbyte hard disc.

Workstations: 20 IBM PCs with Hercules graphics cards, an 8087 coprocessor, and 512 Kbytes of memory.

Printers: Two dot matrix printers are attached to each fileserver.

Back up: A 1/4 inch tape streamer is part of the console for each network.

Software Specifications:

Network software: Novell Advanced Netware Version 1.0

Software run by students includes Fortran and Turbo Pascal compilers, a locally developed assembly language simulator, mathematical and statistical packages, an editor and other course related software.

The Radiology Network

Hardware specifications:

File server: IBM AT with an Intel 80286 processor, 640K base memory, 512K expansion memory and a 20M internal hard disk. The disk will be replaced by a 70M external hard disk when required.

Work Stations: IBM XT, 256 to 640K base memory, 1 floppy disk drive.

Network: G Net (X.25 option) boards in fileserver and workstations. A Mulitdrop coaxial cable connects the fileserver to each workstation, operating a CSMA/CD protocol at 1.4 Mbits per second.

Software specifications:

Network Software: Novell Netware 86 version 4.61, (awaiting an upgrade to Advanced Netware 86, version 1.0).

Database Software: INFORMIX, version 3.03.03. Single user MS-DOS version, produced by RDS (Relational Database Software), USA.

Configuration:

The Christchurch Hospital installation will eventually have 10 XTs, used as work stations, the AT as a dedicated fileserver with a 70M hard disk (expanding to 140M within 2 years), several (2 or 3) network printers, and one local printer attached to the reception workstation.

Appendix B: Synthetic Job Timing

The system call provided by the C compiler to give the time of day is only accurate to the nearest second, which is not sufficient for timing a short IO job.

DOS provides a software interrupt, subfunction 2Ch of interrupt 21h, which returns the time in hundredths of a second. At first glance, this level of accuracy is ample. However, inspection of the BIOS listings [IBM84] revealed that this clock was updated by a hardware clock interrupt which only ticked 18.2 times a second. The time returned by this software interrupt can therefore only be accurate to ±0.03 seconds. The small variations in the average time of the events being timed would make refinement of this error advantageous.

The following method was therefore employed. The clock function provided by the C compiler is called repeatedly until the second changes. The event being timed is then started, and upon completion the number of time function calls until the second changes is counted. As the number of time function calls possible per second is known, the fraction of a second between termination and the next second change can be calculated. The time to initially call the timing procedure is considered negligible. Trials showed the number of calls per second to be 688. 18% of these samples gave the number of 'ticks' per second as 727, obviously some system or network interrupt delays execution 82% of the time. The average inaccuracy is therefore:

$$\pm (0.82 * \frac{1}{688} + 0.18 * \frac{1}{722 - 688})$$

$$= \pm 0.006 \text{ secs}$$

which is adequate for the required purpose. Variables used to record aggregates of times are all of type 'double' to minimize inaccuracies caused by truncation of a smaller real number when added to a larger total.

Appendix C

Appendix C.1

Trimon was used to produce the following graph of the number of users verses time. 'C' denotes a user logged to the local fileserver, from the remote cluster in the engineering 'Cave'. 'R' denotes a user on a local PC logged to the remote server, and 'L' a user logged to the local server from a local PC.

```
This session started at  7:57:02,  7-27-86
           0    5    1    5    2    5    3
           ----------------------------------------
8:00:02    *LCCC
           *LCCC
           *LCC
           *LCC
           *LCC
           *LCC
           *LCC
           *LCC
           *LLCC
           *LLLLLRCC
           *LLLLLLRC
           *LLLLLLRR
 9:00:01   *LLLLLLLRR
           *LLLLLLLRRC
           *LLLLLLLRRCC
           *LLLLLLLLRRCC
           *LLLLLLLLLLLLLLRRCC
           *LLLLLLLLLLLLLLLRRCCC
           *LLLLLLLLLLLLLLLLRRCCCC
           *LLLLLLLLLLLLLLLLRRCCCC
           .
           .  (fileserver down for backup)
           .
10:10:15   *LLLLLLLLLLLC
           *LLLLLLLLLLLLLRCC
           *LLLLLLLLLLLLLRCCC
           *LLLLLLLLLLLLLRCCCCCC
           *LLLLLLLLLLLLLLLRCCCCCCC
           *LLLLLLLLLLLLLLLLRCCCCCCC
           *LLLLLLLLLLLLLLLLRCCCCCCC
           *LLLLLLLLLLLLLLLLRCCCCCC
11:00:02   *LLLLLLLLLLLLLLLLLLLCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLLCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCC
           *LLLLLLLLLLLLLLLLLCCCCCCCCC
           *LLLLLLLLLLLLLRCCCCCCC
12:00:00   *LLLLLLLLLLLRRCCCC
```

```
          *LLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLRRRRCCCC
          *LLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLLRRCCC
13:00:04  *LLLLLLLLLLLLLLLLRRCC
          *LLLLLLLLLLLLRRRCC
          *LLLLLLLLLRRCC
          *LLLLLLLLLRRCC
          *LLLLLLLLLRRCC
          *LLLLLLLLLLRRCC
          *LLLLLLLLLLLLLRRCC
          *LLLLLLLLLLLLLLRRCC
          *LLLLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLLLRRRC
          *LLLLLLLLLLLLLLLLRRRC
14:00:04  *LLLLLLLLLLLLLLLLRRRC
          *LLLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLLLLRRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLLLLCCCCCCCCCCCCCCCC
15:00:03  *LLLLLLLLLLLLLLLLRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLRRCCCCCCCCCCCCCCC
          *LLLLLLLLLLLRRRCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLRRRCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLRRRCCCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLRRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRRCCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCCCCCCCC
16:00:01  *LLLLLLLLLLLLLLLLRRCCCCCCCCCCC
          *LLLLLLLLLLLLLLLLLRRCCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCC
          *LLLLLLLLLLLLLLLLRRCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCC
          *LLLLLLLLLLLLLLLRRCCCCCCC
          *LLLLLLLLLLLLLLRRCCCCCCC
          *LLLLLLLLLLLLLLLLRRCCCCC
          *LLLLLLLLLLLLLLLLLRRCCCCC
          *LLLLLLLLLLLLLLLLRRCCCCCC
          *LLLLLLLLLLLLLLLLLRRCCCCCC
17:00:02  *LLLLLLLLLLLLLLLLLLRRCCCCC
          *LLLLLLLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLLLLLLRRCCCC
          *LLLLLLLLLLLLLLLLLLLRRCCCCC
```

Appendix C

```
          *LLLLLLLLLLLLLLLLLLRRCCCCCC
          *LLLLLLLLLLLLLLLLLRCCCC
          *LLLLLLLLLLLLLLLLRCCC
          *LLLLLLLLLLLLLLLRCC
          *LLLLLLLLLLLLLRCC
          *LLLLLLLLLLLLLLCC
          *LLLLLLLLLLLLLLCC
          *LLLLLLLLLLRCC
18:00:01  *LLLLLLLLLRCCC
          *LLLLLLLLLLRCCC
          *LLLLLLLLLRCCC
          *LLLLLLLLLRCCCC
          *LLLLLLLLLLRCCC
          *LLLLLLLLLRCCC
          *LLLLLLLLLRCC
          *LLLLLLLLLLRCC
          *LLLLLLLLLLLRRCC
          *LLLLLLLLLLLLRRCC
          *LLLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLLRRRCC
19:00:04  *LLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLLRRRCC
          *LLLLLLLLLLLLLRRRCCC
          *LLLLLLLLLLLLLLRRRCCC
          *LLLLLLLLLLLLLLLRRRCCCC
          *LLLLLLLLLLLLLLLLLRRRCCCCCCC
          *LLLLLLLLLLLLLLLLLLRRRCCCCCCC
          *LLLLLLLLLLLLLLLLLRRRCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCC
20:00:04  *LLLLLLLLLLLLLLLRRRCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCCCC
          *LLLLLLLLLLLLLLLLRRRCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRCCCCCCCCCCCC
          *LLLLLLLLLLLLLLLRRCCCCCCCCCCCC
          *LLLLLLLLLLLLLLRRCCCCCCCCCCCCC
          *LLLLLLLLLLLLLRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLLRRCCCCCCCCCCCCCC
          *LLLLLLLLLLLRRCCCCCCCCCCCCCCC
          *LLLLLLLLLLRRCCCCCCCCCCCCCCCC
21:00:02  *LLLLLLCCCCCCCCCCCCCCCC
          *LLLLCCCCCCCCCCCCCCCC
          *LLLCCCCCCCCCCCCCCCC
          *LLCCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCC
22:00:03  *LCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCCC
          *LCCCCCCCCCCCCCCC
```

Appendix C

```
            *LCCCCCCCCCCCCCC
            *LCCCCCCCCCCCCCC
            *LCCCCCCCCCCCCCC
            *LCCCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
23:00:03    *LCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
 0:00:03    *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCCC
            *LCCCCCCCCCCC
            *LCCCCCCCCCCC
            *LCCCCCCCCCC
            *LCCCCCCCC
            *LCCCCCCCCC
            *LCCCCCC
 1:00:03    *LCCCC
            *LCCCC
            *LCCCC
            *LCCCCC
            *LCCCCC
            *LCCCCC
            *LCCCCCC
            *LCCCCC
            *LCCCC
            *LCC
            *LCC
            *LCC
 2:00:03    *LCC
            *LCC
            *LCC
            *LCC
```

Appendix C

Appendix C.2

Results of remote response ratio vs number of users are listed below. The columns are: the number of users, the number of samples for that number of users, the sum of those samples, the sum of the samples squared, the average sample, the variance and the remote response ratio.

Before the System Change

1	164	295.577026	541.895081	1.802299	0.055957	1.000000
2	428	776.755798	1436.815918	1.814850	0.063366	1.006964
3	250	455.204956	854.004150	1.820820	0.100632	1.010276
4	262	494.183140	963.929603	1.886195	0.121388	1.046550
5	7	14.162791	29.223194	2.023256	0.081178	1.122597
6	14	33.446220	81.966400	2.389016	0.147347	1.325538
7	122	253.495636	571.917603	2.077833	0.370459	1.152879
8	276	495.729645	942.437988	1.796122	0.188577	0.996573
9	238	449.782013	898.174255	1.889840	0.202345	1.048572
10	158	328.805237	731.479431	2.081046	0.298865	1.154662
11	703	1323.088623	2613.430908	1.882061	0.175388	1.044256
12	630	1209.112061	2431.027832	1.919225	0.175348	1.064876
13	527	1012.747070	2039.872314	1.921721	0.177713	1.066261
14	188	374.518890	789.511414	1.992122	0.230980	1.105323
15	271	536.752930	1111.066895	1.980638	0.176950	1.098951
16	142	334.239838	823.500305	2.353802	0.258916	1.306000
17	68	142.966568	337.845612	2.102450	0.548024	1.166538
18	81	183.901169	461.623932	2.270385	0.544414	1.259716
19	146	298.401154	668.127075	2.043844	0.398917	1.134020
20	105	218.328491	482.633514	2.079319	0.272942	1.153704
21	176	333.613373	684.132996	1.895531	0.294083	1.051729
22	33	77.218025	191.381271	2.339940	0.324113	1.298309
23	84	193.751450	490.588196	2.306565	0.520094	1.279790
24	58	122.845932	274.813019	2.118033	0.252090	1.175184
25	74	157.709305	365.482178	2.131207	0.396906	1.182494
26	58	129.716568	305.415771	2.236493	0.263890	1.240911
27	155	348.860443	829.678894	2.250713	0.287060	1.248801
28	115	261.914246	629.219666	2.277515	0.284400	1.263672
29	7	15.632268	36.773163	2.233181	0.266211	1.239074
30	28	63.595928	152.230469	2.271283	0.278075	1.260214
31	13	44.690407	180.869843	3.437724	2.095121	1.907410
32	34	92.675873	266.054535	2.725761	0.395361	1.512380
33	70	176.978195	475.646912	2.528260	0.402858	1.402797
34	197	478.664246	1247.576172	2.429768	0.429103	1.348149

After the System Change

1	224	158.645355	112.598778	0.708238	0.001072	1.000000
2	331	234.140991	165.972351	0.707375	0.001048	0.998781
3	8072	5732.290039	4083.177979	0.710145	0.001539	1.002692
4	4461	3257.181641	2383.815918	0.730146	0.001255	1.030933
5	1667	1219.543579	894.920227	0.731580	0.001636	1.032957
6	1615	1170.908447	851.695740	0.725021	0.001711	1.023696
7	1684	1211.875000	875.372009	0.719641	0.001934	1.016100
8	2366	1711.066832	1244.424031	0.723190	0.002958	1.021111
9	2058	1489.164429	1084.055973	0.723598	0.003158	1.021687
10	1551	1124.389546	819.365306	0.724945	0.002737	1.023589
11	979	715.974933	529.746414	0.731333	0.006262	1.032609
12	3291	2425.266533	1799.550118	0.736939	0.003730	1.040524
13	2802	2071.678799	1544.329899	0.739357	0.004504	1.043939
14	2433	1827.193251	1387.963674	0.751004	0.006467	1.060384
15	2703	2037.167111	1556.117666	0.753669	0.007683	1.064146
16	1755	1305.375088	979.426769	0.743803	0.004834	1.050217
17	2037	1539.995634	1178.304614	0.756012	0.006897	1.067454
18	1202	901.553763	684.687351	0.750045	0.007056	1.059029
19	2302	1744.188882	1338.013256	0.757684	0.007154	1.069815
20	2448	1862.822754	1438.839844	0.760957	0.008706	1.074437
21	2183	1681.181553	1316.274893	0.770124	0.009874	1.087380
22	2086	1610.197737	1263.527644	0.771907	0.009878	1.089897
23	1839	1433.790030	1137.506563	0.779657	0.010681	1.100841
24	1931	1509.739777	1201.939985	0.781843	0.011165	1.103927
25	1468	1149.236887	916.377164	0.782859	0.011367	1.105361
26	1495	1178.107597	948.767420	0.788032	0.013633	1.112665
27	1427	1126.697674	910.443327	0.789557	0.014612	1.114818
28	916	709.613339	560.657458	0.774687	0.011931	1.093823
29	1440	1125.845926	893.796568	0.781837	0.009422	1.103919
30	1456	1149.401212	922.843279	0.789424	0.010631	1.114631
31	1031	818.408446	660.377152	0.793801	0.010402	1.120810
32	759	601.687506	485.903053	0.792737	0.011756	1.119309
33	703	565.874998	467.555431	0.804943	0.017153	1.136543
34	227	183.270351	150.885272	0.807358	0.012865	1.139953
35	36	29.572674	24.577246	0.821463	0.007900	1.159869

Appendix C.3

The results of measuring the Undergraduate Network with Netmon are listed below, with the results of each user session totalled for each number of users logged on, when that user logged off. The columns are: the number of users, the number of user sessions sampled, the remote component ratio, the total number of local samples and the total number of remote samples.

users	samples	ratio		
1	26	0.017049	1063347	18443
2	35	0.015507	1297583	20438
3	65	0.028641	2112294	62282
4	99	0.014275	3295822	47729
5	94	0.017931	3605366	65828
6	78	0.025849	2474111	65649
7	99	0.019578	3690304	73693
8	99	0.022311	2934821	66972
9	105	0.018354	3772939	70542
10	89	0.018521	2547383	48071
11	77	0.021866	2521429	56367
12	91	0.019924	3337708	67852
13	81	0.023165	2732539	64800
14	75	0.017052	3226135	55965
15	66	0.018990	2905212	56239
16	71	0.019691	2438334	48978
17	58	0.028143	1930176	55894
18	40	0.016428	1406304	23488
19	30	0.019767	1099734	22177
20	19	0.026031	476263	12729

Appendix C.4

The results collected using Netmon are listed below sorted by remote response ratio for each user for whom more than 5 logged hours were recorded.

ratio	user	samples	hours
0.003443	308	11	17.100900
0.004344	305	5	6.454670
0.005957	1271	10	10.643071
0.007100	1180	4	5.498474
0.008517	1415	8	6.055174
0.008524	1288	8	8.788004
0.008993	1143	9	5.339270
0.010072	1462	10	8.214545
0.010330	1312	28	30.917170
0.010563	1230	9	6.201267
0.011160	1233	9	8.357799
0.011437	1367	34	15.197176
0.011460	1130	18	14.848825
0.012646	1234	9	6.416880
0.012866	1194	14	7.091468
0.013185	1177	10	7.065705
0.013434	1351	9	6.947115
0.013874	1098	13	8.193254
0.014057	317	16	8.968132
0.014305	1128	10	8.272772
0.014388	1157	16	8.824802
0.014407	1172	12	7.512088
0.014863	1092	13	7.416941
0.015045	1384	10	8.950733
0.015741	1062	23	12.053297
0.015775	1035	12	6.410592
0.015912	1499	13	7.620589
0.016328	1174	33	28.545437
0.018683	1255	26	14.969780
0.019569	1145	13	7.957738
0.020269	1135	30	19.546047
0.020786	1199	55	33.768620
0.022087	1247	26	16.211233
0.022418	1151	10	10.627564
0.022607	1362	25	9.607875
0.022854	1349	10	7.551267
0.022929	1421	10	5.436905
0.023646	1213	17	9.411920
0.024040	1072	14	5.870177
0.024192	1357	11	5.179029
0.026181	1428	43	12.456090
0.027716	1074	31	12.372054
0.028462	1099	42	11.487378
0.045569	1012	25	9.716117
0.048763	1015	17	15.382830
0.050593	1013	36	19.766285
0.103272	1021	18	9.236584

Appendix D

The program code described in section 4 is listed in this appendix.

D.1 Trimon.c

```
/*  A network monitor, written by David Tripp, UOC, May 1986.

    Invoked with command:  trimon [-mup]

       flags : none  will do both IO and user monitoring
               m     will do IO monitoring
               u     will keep user log
               p     will also print user log to screen

    The IO monitoring is done by repeatedly writting a file, and
    timing this operation, tabling each time against the number
    of users logged on at the point in time.

    The user monitor averages the number of users of a five
    minute period and records this in the given log file.  It
    records the number of users logged on locally, those logged
    to a remote server from this one, and those logged from a
    remote server to this one.
*/

#define NUM_STATION 50
#define NUM_PLUS1   51
#include <stdio.h>

double total[NUM_PLUS1],  /* iotime for each number of users  */
    squared[NUM_PLUS1];/* iotime squared for each no.of users */
int calls[NUM_PLUS1];     /* samples for this number of users */
int ioflag, userflag, pflag;

setup(resultfile, userfile)
char *resultfile, *userfile;
   {
   FILE *fopen(), *fp;
   int i, call;
   float time, square, sd, mean, rr;
   char now[9], today[9];

   /* initalize the tables for times of IO tests */
   for (i=0; i<=NUM_STATION; i++)
      calls[i] = total[i] = squared[i] = 0;

   /* if doing IO test, then read in previous results, if any */
   if (ioflag)
      {
      printf("\nEnter filename for IO results: ");
```

```
      scanf("%s", resultfile);
      if ((fp = fopen(resultfile, "r")) != NULL)
         {
         while (fscanf(fp, "%d %d %f %f %f %f %f\n", &i, &call,
                        &time, &square, &mean, &sd, &rr) != EOF)
            {
            calls[i] = call;
            total[i] = time;
            squared[i] = square;
            }
         close(fp);
         }
      }

   /* write user log heading */
   if (userflag)
      {
      printf("\nEnter filename for User log: ");
      scanf("%s", userfile);

      if ((fp = fopen(userfile, "a")) == NULL)
         {
         printf("Can't open user log file\n");
         exit(1);
         }
      times(now);
      dates(today);
      fprintf(fp, "\nThis session started at %s, %s\n", now,
                   today);
      fprintf(fp, "         0    5    1    5    2    5    3\n");
      fprintf(fp, " ---------------------------------------\n");
      close(fp);
      }

    /* print userlog to screen */
    if (pflag)
       {
       times(now);
       dates(today);
       printf("\nThis session started at %s, %s\n", now, today);
       printf("          0    5    1    5    2    5    3\n");
       printf("  ---------------------------------------\n");
       }
   }


/* get # of users by getting login table for each workstation */
nousers(local, remote, cave)
int *local, *remote, *cave;
   {
   char req[4], reply[260];
   int i, error, count;
```

```
   *local = 0;
   *remote = 0;
   *cave = 0;
   count = 0;
   for ( i = 1, count = 0; i <= NUM_STATION; i++)
      {
      req[0] = 4;     /* length of buffer excluding this word? */
      req[1] = 0;
      req[2] = 5;     /* get terminal info */
      req[3] = i;     /* of station i      */

      reply[0] = 4    /* size of reply buffer */
      reply[1] = 1;

      error = sysreq(227, reply, req);
      if (!error)
         {
         if (*(reply + 2))    /* that station logged on */
            {
            count++;
            if (*(reply + 2) == 'R')
               (*remote)++; /* logged on as remote dummy */
            else
               if (i<= 25)  /* station logged to local server */
                  (*local)++;
               else
                  (*cave)++; /* logged on from remote server */
            }
         }
      else
        printf("station %d returned %d\n", i, error);
      }
   return(count);
   }

/* execute network function call n, reply and req point to
   packets to be sent and received */
sysreq(n, reply, req)
int n;
char *reply, *req;
  {
#asm
  mov  ah,[bp+4]      ; function number
  push ds
  pop  es             ; put ds into es
  mov  si,[bp+8]      ; pointer to request buffer
  mov  di,[bp+6]      ; pointer to reply buffer
  int  21h
  mov  ah,0           ; zero top half, result in bottom half
#
  }
```

```
/* count number of calls to time ('ticks') till second change */
second(buf, count)
char *buf;
int *count;
   {
   char s1, s2;

   times(buf);
   s1 = buf[6];
   s2 = buf[7];
   *count = 1;

   while (s1 == buf[6] && s2 == buf[7])
      {
      times(buf);
      (*count)++;
      }
   }


/* turn a string in format hh/mm/ss into integer no. of secs
   since midnight */
unsigned secs(buf)
char buf[];
   {
   int t;

   t = (buf[6] - '0')*10 + (buf[7] - '0');
   t = t + (buf[3] - '0')*600 + (buf[4] - '0')*60;
   t = t + (buf[1] - '0')*3600 + (buf[1] - '0')*36000;

   return(t);
   }


/* create the files te1, ..te5 and write 100 chars to each */
run()
   {
   FILE *fp, *fopen(), *fclose();
   char filename[4];
   int i, j;

   filename[0] = 't';
   filename[1] = 'e';
   filename[3] = '\0';

   for (j = 0; j < 9; j++)
      {
      filename[2] = j%10 + '0';
      fp = fopen(filename, "w");
      for (i = 0; i < 100; i++)
         putc('c', fp);
      fclose(fp);
```

```
        }
    }


/* do an IO run and time it */
float iotime(num)
int num;
    {
    int count;
    char buf1[9], buf2[9], now;
    int sec1, sec2;
    float io;
    unsigned secs();

    second(buf1, &count);
    run();
    second(buf2, &count);

    sec1 = secs(buf1);  sec2 = secs(buf2);
    if (sec2 - sec1 > 100 || sec1 > sec2)
        {
        times(now);
        printf("Trimon: daft io time: %d secs,  %d users at %s\n",
                sec1 - sec2, num, now);
        return(total[num]/(float)calls[num]); /* mean so far */
        }
    return(sec2 - sec1 - ((float)count/688.0)) ;
    }


/* copy string from s1 to s2 */
copy(s1, s2)
char *s1, *s2;
    {
    while (*s2++ = *s1++);
    }


main(argc, argv)
int argc;
char *argv[];
  {
  float iotime(), io;
  double mean, meansq, mean1;
  int num, i, j, local, remote, cave, totloc, totrem, totcave,
      call;
  char now[9], old[9], userfile[12], resultfile[12],
       userout[60], *p;
  FILE *fopen(), *fp;

  /* get arguments and set flags */
   if (argc == 2 && (*++argv)[0] == '-')
      {
```

```
   ioflag = 0;
   userflag = 0;
   pflag = 0;
   i = 1;
   while ( (*argv)[i] )
      switch ( (*argv)[i++] )
         {
         case 'm' : ioflag = 1;
                    break;
         case 'u' : userflag = 1;
                    break;
         case 'p' : pflag = 1;
                    break;
         default  : printf("Trimon: invalid flag\n");
                    exit(1);
         }
   }
else
   {
   ioflag = 1;
   userflag = 1;
   pflag = 0;
   }

setup(resultfile, userfile);
totloc = totrem = totcave = call = 0;
times(old);

/* do lots of iterations! */
for (i=0; i<32000; i++)
   {
   num = nousers(&local, &remote, &cave);
   if (ioflag)
      {
      io = iotime(num);
      calls[num]++;
      total[num] += io;
      squared[num] += io*io;
      }

   totloc += local;
   totrem += remote;
   totcave += cave;
   call++;
   times(now);
   if (!((now[4] - '0')%5) && (old[4] - '0')%5 && (userflag
         || pflag))
      {
      /* every five minutes output user log and reset */
      p = userout;
      if (now[3] == '0' && old[3] != '0')
         copy(now, p);
      else
```

```
            copy("          ", p);
        copy("  *", p+8);
        p = p+11;
        for (j=1; j<=totloc/call; j++)
            *p++ = 'L';
        for (j=1; j<=totrem/call; j++)
            *p++ = 'R';
        for (j=1; j<=totcave/call; j++)
            *p++ = 'C';
        *p++ = '\n';
        *p = 0;
        if (userflag)
            {
            fp = fopen(userfile, "a");
            fprintf(fp, "%s", userout);
            close(fp);
            }
        if (pflag)
            printf("%s", userout);
        else
            printf("    Trimon: updated user log at %s\n", now);
        totloc = totrem = totcave = call = 0;
        }
    copy(now, old);

    second(now, &j);
    second(now, &j);
    second(now, &j);
    if (!ioflag)
        {
        second(now, &j);
        second(now, &j);
        }

    if (i%100 == 0 && ioflag)
        {
        /* every 100 loops dump IO results in case of crash */
        fp = fopen(resultfile, "w");

        if (calls[1])
            mean1 = total[1]/(double)calls[1];
        else
            mean1 = 1;

        for (j=1; j<=NUM_STATION; j++)
            if (calls[j])
                {
                mean = total[j]/(double)calls[j];
                meansq = squared[j]/(double)calls[j];
                fprintf(fp,"%d %d %f %f %f %f %f\n", j, calls[j],
                  total[j], squared[j], mean, meansq - mean*mean,
                  mean/mean1);
                }
```

```
        close(fp);
        printf("    Trimon: updated IO results at %s\n", now);
        }
    }
}
```

D.2 Netmon

The 4 programs that constitute the Netmon suite are listed below, as discussed in section 4.2. Listing D.2.5 is the assembler code written to print the stack during normal workstation operation. From this screen output the value of the CS register, when the workstation was in remote state, was found.

D.2.1 Netmon.asm

This code installs the state sampling monitor in the workstation. It is executed from the fileservers autoexec.bat file.

```
data    segment
        org  6ch
data ends

cseg    segment
        assume cs:cseg,ds:data
        org    100h
start:  mov    dx,5ch     ;set ds:dx to loc 5c in program prefix
        mov    al,1ch     ;take over interrupt 1ch
        mov    ah,25h
        int    21h
        mov    di,dx      ;move monitor to 5ch in prefix
        mov    si,monadr
        mov    cx,monlen
        rep    movsb
        xor    ax, ax     ;zero the words used to store counters
        mov    es, ax
        mov    es:[130h], ax
        mov    es:[132h], ax
        mov    es:[134h], ax
        mov    dx,di      ;end program but leave program resident
        int    27h

mon:    sti
        push   bx         ;save extra register used
        push   cx         ; (BIOS already saved ax,dx, ds)
        push   bp
        push   es

        xor    ax, ax
        mov    es, ax
        mov    bp, sp
        mov    ax, ss:[bp + 22] ;get old CS off stack

        cmp    ax, 1547
        je     remote
local:  mov    ax, es:[130h]    ;increment local counter
```

```
        cmp     ax, 65535
        je      over
        inc     ax
        mov     es:[130h], ax
        jmp     end
over:   mov     word ptr es:[130h],0;local counter has overflowed
        mov     ax, es:[132h]
        inc     ax
        mov     es:[132h], ax
        jmp     end
remote: mov     ax, es:[134h]     ;increment remote counter
        inc     ax
        mov     es:[134h], ax

end:    pop     es    ;restore registers to their pre-call values
        pop     bp
        pop     cx
        pop     bx
        iret
monend:
cseg    ends

monadr  equ     (offset mon - offset start) + 100h
monlen  equ     offset monend - offset mon
        end     start
```

D.2.2 Zero.asm

This code initialises the state counters in the workstations memory to 0. It is executed from the users login script.

```
; code to set the counters in memory to zero

cseg    segment
        assume cs:cseg
        org 100h

start:  xor     ax, ax
        mov     es, ax
        mov     si, 130h
        mov     es:[si], ax
        mov     es:[si + 2], ax
        mov     es:[si + 4], ax
        ret

cseg    ends
        end     start
```

D.2.3 Netread.c

The following program reads the values of the state counters from low memory, and writes them to the result file, with other user information.

```
/*  A network monitor, written by David Tripp, UOC, August 1986.

    This code is invoked when the user logs off. At that point it
    reads the counters maintained during a session by netmon,
    invoked by the autoexec startup file, and appends these to
    the results file, netmon.res.

    The counts are stored in memory as follows:
      0:134h  number of samples where workstation waiting on the
              network, when the CS register is 1547(assumed to
              be the segment of the packet processing code in the
              PC with no local disc drives?, configuration
              dependent)
      0:130h  number of samples when not in the above state
      0:132h  overflow from 0:132h

    The output format is:
              number of users at the time this user logged off
              overflow
              local
              remote
              station number
              logoff time
              usercode
*/

#define NUM_STATION 24    /* local stations only */
#include <stdio.h>


/* get # of users by getting login table for each workstation */
nousers()
   {
   char req[4], reply[260];
   int i, error, count;

   count = 0;
   for ( i = 1, count = 0; i <= NUM_STATION; i++)
      {
      req[0] = 4;     /* length of buffer excluding this word? */
      req[1] = 0;
      req[2] = 5;     /* get terminal info */
      req[3] = i;     /* of station i      */

      reply[0] = 4;  /* size of reply buffer */
      reply[1] = 1;
```

```
      if (!error)
         {
         if (*(reply + 2))    /* that station logged on */
            count++;
         }
      else
        count = 99;/* a rediculous value=>ignore these results */
      }
   return(count);
   }


/* execute a system function call.  The request is given in req and
   the reply returned in reply */

sysreq(n, reply, req)
int n;
char *reply, *req;
  {
#asm
  mov  ah,[bp+4]      ; function number
  push ds
  pop  es             ; put ds into es
  mov  si,[bp+8]      ; pointer to request buffer
  mov  di,[bp+6]      ; pointer to reply buffer
  int  21h
  mov  ah,0           ; zero top half, result in bottom half
#
  }


/* read counters maintained by netmon out of absolute memory */

getvalues(local, overflow, remote)
unsigned *local, *overflow, *remote;
  {
#asm
  xor  ax,ax
  mov  es,ax
  push si

  mov  ax, es:[130h]
  mov  si, [bp+4]
  mov  ss:[si], ax

  mov  ax, es:[132h]
  mov  si, [bp+6]
  mov  ss:[si], ax

  mov  ax, es:[134h]
  mov  si, [bp+8]
  mov  ss:[si], ax
  pop  si
```

```
#
  }


/* get this workstations physical connection number */

int wsid()
  {
#asm
  mov  ah,220
  int  21h
  mov  ah,0
#
  }

strcpy(s1, s2)
char *s1, *s2;
  {
  while (*s2++=*s1++);
  }


/* get this users usercode */

getuser(usercode)
char *usercode;
  {
  char reply[260], req[4];

  req[0] = 4;
  req[1] = 0;
  req[2] = 5;
  req[3] = wsid();
  reply[0] = 4;
  reply[1] = 1;

  sysreq(227, reply, req);
  strcpy(reply+2, usercode);
  }

main()
  {
  FILE *fopen(), *fp;
  unsigned local, overflow, remote;
  int users, nousers();
  char usercode[17], now[9];

  getvalues(&local, &overflow, &remote);
  users = nousers();

  getuser(usercode);
  times(now);
```

```
/* open results file and write this users logoff info to it */
fp = fopen("\\cosc460\\monitor\\network\\netmon.res", "a");
fprintf(fp, "%2d %5u %5u %5u %2d %s %s\n", users, overflow,
        local, remote, wsid(), now, usercode);
close(fp);
}
```

D.2.4 Netanal.c

This program reads and processes the results file.

```
/*  Netanal, a program to analyse the results of netmon, as
    written to the file netmon.res by netread.exe when the user
    logs off.

    This code builds a table of the number of local and remote
    samples against the number of users logged on when the user
    logged off.

    The ratio of local to remote for each number of users is
    printed to standard output.
*/

#include <stdio.h>

long local[21], remote[21];
int samples[21];

main()
   {
   unsigned users, over, rem, loc, i;
   char rest[60];
   FILE *fp, *fopen();

   for (i=0; i<=20; i++)
      local[i] = remote[i] = samples[i] = 0;

   fp = fopen("netmon.res", "r");

   /* read in the results from netmon.res */
   while (fscanf(fp, "%2u %u %u %u %s %s %s\n",
        &users, &over, &loc, &rem, &rest, &rest, &rest) != EOF)
      {
      if ( over < 3 && rem > 0)
         {
         /* printf("%2d %5u %5u %5u", users, over, loc, rem); */
         local[users] += (float)(65536*over) + (float)loc;
         remote[users] += (float)rem;
         samples[users]++;
         }
      }

   /* print the results to standard output */
   printf("   users     samples   ratio\n");
   for (i=0; i<=20; i++)
      if (samples[i] != 0)
         printf("  %2d    %3d  %f  %8ld  %8ld\n", i, samples[i],
              ((float)remote[i]/(float)(remote[i] + local[i])),
              local[i], remote[i]);
   }
```

D.2.5 Stackout.asm

This assembler program prints the current nine top cells of the stack at the top of the screen. This was used to find the value of the CS register when the workstation was in remote state.

```
vidram  segment at 0b000h ; Monochrome display RAM origin.   Use
        org   10*2        ;  0b800h for color/graphics adaper
clkcol  label  word       ; stating position for stack display.

cseg    segment
        assume cs:cseg
        org    100h
start:  mov    dx,5ch     ;set ds:dx to loc 5c in program prefix
        mov    al,1ch     ;take over interrupt 1ch
        mov    ah,25h
        int    21h
        mov    di,dx      ;move monitor to 5ch in prefix
        mov    si,clkadr
        mov    cx,clklen
        rep    movsb
        mov    dx,di      ;end program, leave program resident
        int    27h

clock:  sti
        push   bx         ;save extra register used
        push   cx         ; (BIOS already saved ax,dx, ds)
        push   bp

        assume ds:vidram
        mov    ax,vidram  ;establish video RAM base address
        mov    ds,ax
        lea    bx,clkcol  ;point at start of screen display
        call   blank

        mov    bp, sp
        mov    ax, [ss:bp + 22]
        call   print
        call   colon
        mov    ax, [ss:bp + 20]
        call   print
        call   colon
        mov    ax, [ss:bp + 18]
        call   print
        call   colon
        mov    ax, [ss:bp + 16]
        call   print
        call   colon
        mov    ax, [ss:bp + 14]
        call   print
        call   colon
        mov    ax, [ss:bp + 12]
```

```
        call    print
        call    colon
        mov     ax, [ss:bp + 10]
        call    print
        call    colon
        mov     ax, [ss:bp + 8]
        call    print
        call    colon
        mov     ax, [ss:bp + 6]
        call    print
        call    colon
        pop     bp
        pop     cx
        pop     bx
        iret

print : mov     dx, 0       ;routine to print ax in decimal
        mov     cx, 10
        div     cx          ;divide ax by 10, remainder in dx
        push    dx          ;push dx (last digit of number)

        mov     dx, 0
        div     cx
        push    dx

        mov     dx, 0
        div     cx
        push    dx

        mov     dx, 0
        div     cx
        push    dx

        mov     dx, 0
        div     cx
        push    dx

        pop     ax          ;pop into ax and print the character
        call    disply
        pop     ax
        call    disply
        pop     ax
        call    disply
        pop     ax
        call    disply
        pop     ax
        call    disply
        ret

disply: cbw                 ;display al at vidram = ds:[bx]
        add     al,30h      ;convert to acsii
vstor1: mov     [bx],al
        inc     bx
```

```
        mov     byte ptr [bx],7
        inc     bx
        ret
blank:  mov     al," "
        jmp     vstor1
colon:  mov     al,":"
        jmp     vstor1
clkend:
cseg    ends

clkadr  equ     (offset clock - offset start) + 100h
clklen  equ     offset clkend - offset clock
        end     start
```

And now we can all get some sleep . . . .

[Shakespeare1613]